VZ-LINK MAGAZINE

NEW ZEALAND

VZ 300/200

COMPUTER USERS CLUB

ISSUE 32

JUNE 89

COMMENT

It's that time of month again.
This issue is on time for a change.I have been very busy at work and have had little time for my computer.

With the Winter now with us,I hope we all get the time to exercise the fingers on the VZ.
I would still love to receive programs and items for this newsletter.Think about it,Maybe you have an idea for a game or simple program.

I have had offers of help from various people,as far as compiling the newsletter etc is concerned.THANK YOU,I will be contacting you in the near future.

THANK YOU,once again to those that made the effort to attend the workshop.
Also Thanks to My wife and the other ladies that supply the lunch and drinks.

This issue should have something of interest to everyone. However I would like to point out that I have heaps of other Information,some which I can not use in the newsletter.
If you need help or are interested in a subject not yet covered in these newsletters,Please feel free to Ring me.

PHONE 761-451

Well,until next issue Have a good time and remember those items for the newsletter.

Regards Peter

BLOCK TRANSFERS

BLOCK TRANSFERS FROM BASIC by Chris Hobrough.

Many VZ'ers who read my recent article on moving blocks of memory around may like to take advantage of the techniques but not have sufficient experience with machine language to do so. Well here is a program which will allow you to set up and call a m/l block transfer routine from BASIC.
The routine is the final version from the article, as it's the most versatile. It's 28 bytes long and sits in a protected area at the top of RAM. In addition, it uses 7 bytes of free space in the comms region; 3 pairs of bytes which hold the parameters for the routine (FROM, TO & COUNT) and one which is used as a flag so that memory is not lowered a bit more each time you run the program.
The first section lowers the Top of Memory to make room for the routine and will work for any size system. Then the routine itself is loaded into this safe area. The next section, from 1000 to 1600 is for demonstration only and consists of a "flashy" intro to show what can be done and a user input piece which accepts the parameters to move any block you wish. Of course you will have to put something in there to move first and also a routine to PEEK at the results. Finally, there's the routine which passes the parameters for the move to the transfer routine and then calls it.
You can adapt this for your own needs by replacing 1000 to 1600 with your own program. It must define X (FROM), Y (TO) and Z (COUNT), and then GOSUB 2000. The simplest thing to do the first time around is to move the prompt lines around the screen. Remember TEXT RAM starts at 28672 and ends at 29183.

```
1 '************************
2 '*                      *
3 '*  BLOCK TRANSFER DEMO  *
4 '*                      *
5 '*   BY CHRIS HOBROUGH   *
6 '*        DEC '87        *
7 '************************
17 '
18 'LOWER T.O.M.
19 '
20 IFPEEK(30797)=128THEN100
30 TM=PEEK(30897)+256*PEEK(30898)-28
40 POKE30897,TM-256*INT(TM/256)
50 POKE30898,INT(TM/256)
60 CLEAR 100
70 POKE30797,128
97 '
98 'LOAD TRNSF ROUTINE
99 '
100 TM=PEEK(30897)+256*PEEK(30898)
110 OG=TM+1
120 FORA=1TO28
130  READB
140  POKETM-65536+A,B
150 NEXT
160 AD=TM+26
170 POKETM-65536+20,AD-256*INT(AD/256)
180 POKETM-65536+21,INT(AD/256)
199 '
200 DATA 42,71,120              :'TRNSF LD   HL,(FROM)
210 DATA 237,91,73,120          :'      LD   DE,(TO)
220 DATA 237,75,75,120          :'      LD   BC,(COUNT)
230 DATA 55                     :'      SCF
240 DATA 63                     :'      CCF
250 DATA 237,82                 :'      SBC  HL,DE
260 DATA 42,71,120              :'      LD   HL,(FROM)
270 DATA 250,0,0                :'      JP   M,MVUP
280 DATA 237,176                :'      LDIR
290 DATA 24,2                   :'      JR   END
300 DATA 237,184                :'MVUP  LDDR
310 DATA 201                    :'END   RET
997 '
998 'MAIN PROGRAM
999 '
1000 CLS:PRINT@199,"BLOCK TRANSFER DEMO"
1010 PRINT@267,"PLEASE WAIT"
1017 '
1018 'LOAD PATTERN INTO VIDEO RAM
1019 '
1020 PK=29184
1030 FORX=0TO255
1040  POKEPK,128
1050  PK=PK+1
1060 NEXTX
1070 CL=143:D=1:F=1
1080 FORI=1TO2
1090  FORJ=1TO4
1100   FORK=1TO4
1110    FORX=1TO8
1120     POKEPK,CL
1130     TC=CL
1140     CL=CL+16*D
1150     IFCL>255THENCL=143
```

```
1160      IFCL<143THENCL=255
1170      PK=PK+1
1180     NEXTX
1190     CL=TC
1200     D=D*(-1)
1210    NEXTK
1220    CL=CL+16*F
1230    IFCL>255THENCL=143
1240   NEXTJ
1250   F=F*(-1)
1260 NEXTI
1270 FORX=0TO255
1280  POKEPK,128
1290  PK=PK+1
1300 NEXTX
1317 '
1318 'FLASH PATTERN
1319 '
1320 Y=28672:Z=512
1330 FORI=1TO10
1340  X=29184
1350  GOSUB2000
1360  X=29440
1370  GOSUB2000
1380 NEXTI
1390 GOSUB3000
1497 '
1498 'USER INPUT
1499 '
1500 CLS:PRINT
1510 INPUT"SOURCE ADDRESS (FROM)";X
1530 INPUT"DESTINATION ADDRESS (TO)";Y
1550 INPUT"NUMBER OF BYTES (COUNT)";Z
1570 GOSUB2000
1580 GOSUB3000
1590 GOTO1500
1600 END
1997 '
1998 'MOVE Z BYTES FROM X TO Y
1999 '
2000 POKE30791,X-256*INT(X/256)  'FROM-LOW BYTE
2010 POKE30792,INT(X/256)        'FROM-HIGH BYTE
2020 POKE30793,Y-256*INT(Y/256)  'TO-LOW BYTE
2030 POKE30794,INT(Y/256)        'TO-HIGH BYTE
2040 POKE30795,Z-256*INT(Z/256)  'COUNT-LOW BYTE
2050 POKE30796,INT(Z/256)        'COUNT-HIGH BYTE
2070 POKE30862,OG-256*INT(OG/256)
2079                             'SET USR POINTERS
2080 POKE30863,INT(OG/256)
2090 C=USR(0)                    'CALL TRNSF
2100 RETURN
2997 '
2998 'WAIT
2999 '
3000 PRINT@483,"PRESS <RETURN> TO CONTINUE";
3010 CH$=INKEY$:CH$=INKEY$
3020 IFCH$<>CHR$(13)THEN3010
3030 RETURN
```

SPLIT SCREEN

MIXED GRAPHICS / TEXT FOR THE VZ !

If you're ever tried to write a program which involves graphics and text on the VZ, you'll know how frustrating it is to have to switch back and forth between mode 0 and mode 1, and having to redraw your graphics each time. This months assembly routine overcomes this problem by achieving what has always seemed the impossible - displaying text and hi-res graphics at the same time!

This article was inspired by Larry Taylor, who has produced some excellent games using the technique. Thanks to Larry for his assistance in getting the routine going.

The small program is yet another interrupt-driven routine. If you are interested in how it works, read the 'How it Works' segment below. Only the assembly language version is given here, so you'll need an Editor Assembler of some kind. The program automatically locates itself at the top of available memory. As soon as the program is loaded & run, it resets the top of memory pointer, so the system is not aware of its presence. The split-screen effect is then turned on with the X=USR(0) command, and turned off with a single POKE statement. A demonstration program is included to show how to use the split-screen from BASIC. As written here, the program gives four lines of text at the top of the screen, and hi-res graphics for the rest of the screen (128 columns x 48 rows).

Firstly, fire up your Editor and type in the program as shown in the listing. Set the origin to 31465 (the start of user RAM) using the command O31465. Note that the values in lines 41 & 42 are DIFFERENT FOR A VZ200 AND VZ300. Insert the pair of values appropriate for your machine.* Now assemble the program (enter A), and correct any errors if they occur. Save the source code now, just in case you've made an error and the system crashes! Save the object code next.

Switch off and CRUN or BRUN (if you have a disk EDASM) the object code. After loading, the READY message should appear. You can now use the split-scrren mode in any program you wish. To switch the split-screen on, use the command X=USR(0). If you want the top four lines of text to be clear, then do a CLS BEFORE selecting the split-screen.

Now that you're in the mixed text/graphics mode, there are a few rules we need to point out. Firstly, DO NOT PRESS CTRL-BREAK when in the split mode, or you will lose control of the system. Also, DO NOT USE ANY INPUT STATEMENTS while in the mode, or the same will occur. The keyboard scanning routine is disabled when in split-screen mode, to avoid bad flicker. I recommend that you use only PRINT@ statements, when printing, and INKEY$ statements if you want user input. All the graphics commands still work OK (except of course MODE(1)), but remember that the size of the display has been reduced to 128 x 48 (the Y-axis has values rangeing from 4 to 51 inclusive).

To get back to text mode, use the commands:

```
POKE 30845,201: POKE 26624,0: CLS
```

BREAK and INPUT will then be back to normal.

* N.B 2 extra lines are added between lines 45 + 46 for a VZ300.

REPRINTED WITH PERMISSION OF MARK HARWOOD FROM VZ USER MAGAZINE
AUSTRALIA

```
001        LD   SP,7700H       ;Set a low stack while we set up
002        LD   HL,(78B1H)     ;Get top of memory
003        LD   BC,END-STRT    ;Get length of program
004        INC  BC
005        PUSH BC             ;& save it
006        XOR  A              ;Clear flags
007        SBC  HL,BC          ;Set aside space for routine
008        LD   (78B1H),HL     ;& lower TOM accordingly
009        PUSH HL             ;& save it
010        XOR  A              ;Clear flags
011        LD   BC,33H         ;Set aside 50 bytes for
012        SBC  HL,BC          ;string space
013        LD   (78A0H),HL
014        POP  DE             ;Get back TOM
015        INC  DE             ;TOM + 1
016        LD   (787EH),DE     ;Set interrupt vector to start
017        PUSH DE             ;& save it
018        LD   HL,ON-STRT     ;Calculate position of 'ON' routine
019        ADD  HL,DE          ;in high memory
020        LD   (30862),HL     ;& set USR vector to it
021        POP  DE             ;Get back new start for routine
022        LD   HL,STRT        ;Get actual start of routine
023        POP  BC             ;Get back length of routine
024        LDIR                ;& block move program to high memory
025        CALL 1B4DH          ;Do a 'NEW'
026        JP   1A19H          ;& back to BASIC
027 STRT   POP  HL             ;Remove RET address from stack
028        JR   SPLT           ;Split screen routine
029 RET    CALL 30E8H          ;Carry out other essential
030        CALL 2EDCH          ;interrupt CALLs as they appear
031        POP  HL             ;in ROM (except keyboard scan)
032        POP  DE             ;and then return
033        POP  BC
034        POP  AF
035        EI
036        RETI
037 SPLT   PUSH AF             ;SPLIT ROUTINE starts here
038        PUSH BC             ;save registers to be used
039        XOR  A              ;A=0
040        LD   (26624),A      ;Select text mode
041        LD   A,237 or 254   ;'Fine delay adjust'=254 for V2300
042 DLY    LD   B,11  or 10    ;'Coarse delay adjust'=10 for V2300
043 LOOP   DJNZ LOOP           ;Delay in text mode for 4 text lines
044        DEC  A              for V2300 insert:  LD B,08
045        JR   NZ,DLY                           LP DJNZ LP in this gap
046        LD   A,8            ;At this point 4 text lines
047        LD   (26624),A      ;have been scanned, so select graphics
048        POP  BC             ;Restore registers
049        POP  AF             ;& return to main routine
050        JR   RET
051 ON     PUSH AF             ;TURN ON SPLIT SCREEN routine
052        PUSH BC             ;(Called by BASIC's USR command)
053        PUSH DE             ;Save registers to be used
054        PUSH HL
055        DI                  ;Disable interrupt to set up
056        LD   A,195          ;Set interrupt vector to jump to
057        LD   (30845),A      ;our interrupt routine (STRT)
058        LD   HL,28800       ;Start of graphics area to be
059        LD   (HL),0         ;blanked with zeroes
```

```
060        LD    DE,28801
061        LD    BC,1600        ;No of bytes to clear
062        LDIR                 ;Clear graphics area to be displayed
063        LD    A,32           ;Switch off screen blanking
064        LD    (30779),A      ;whenever a PRINT occurs
065        POP   HL             ;Get back registers
066        POP   DE
067        POP   BC
068        POP   AF
069        EI                   ;Enable interrupt
070 END    RET                  ;& return.
```

DEMONSTRATION BASIC PROGRAM :

```
10 CLS:'       CLEAR 4 LINES OF TEXT
20 X=USR(0):' SWITCH ON SPLIT SCREEN
36 PRINT@5,"THIS IS A DEMONSTRATION"
38 PRINT@37,"OF SPLIT-SCREEN DISPLAY"
39 PRINT@64,"DONT HIT BREAK WHEN SCREEN SPLIT"
40 FOR Z=0 TO 1000
42 COLOR RND(4): SET(RND(127),3+RND(48))
50 NEXT
60 POKE 30845,201:' SWITCH OFF SPLIT SCREEN
70 POKE 26624,0: CLS:' SET TEXT MODE & CLEAR SCREEN
```

HOW IT WORKS :

The routine is effectively an interrupt routine which is executed at the start of each screen scan. Firstly, the screen is set to text mode, and then a time-critical delay is used to wait until 4 text lines have been displayed. The delay is in the form of two nested loops, one having a 'coarse' time adjustment, the other a 'fine' adjustment. The values of each differ between the VZ200 and VZ300 since the two machines have differing clock speeds. If your machine's clock speed has been altered, then you may need to adjust one or both of these values slightly to obtain a stable display. After the delay is complete, the screen is set to graphics mode for the remainder of the screen scan. Note that since the routine requires no keyboard scan (which would cause flicker) the remainder of interrupt routines are CALLed from the routine instead of from the usual ROM locations 2EB8H etc.

The routine at the start of the program simply moves this routine to the top of memory and resets all pointers etc. The 'ON' routine is called from the USR command, and selects the interrupt routine, clears a portion of the graphics, display area, and 'tricks' the system into thinking it's still in text mode by setting the copy of the output latch appropriately. This prevents text mode being selected when a PRINT statement is executed.

It would be possible to have more text at the top of the screen by having a longer delay, and blanking less graphics RAM, but remember that BASIC is slowed down to a fairly great extent.

VECTORS & INTERRUPTS - EXPLAINED.

by Bob Kitch.

In a recent VZ newsletter a Letter-To-The-Editor sought information on the use of the word "vector" in computer literature. The Editor provided a dictionary definition of the term, however a "de-jargonized" definition is herein provided. A practical program showing how to steal vectors to make an interrupt driven real time clock on the VZ is detailed.

DEFINITIONS.

The word vector has a number of common useages - all roughly meaning the same - but having slightly different implications according to the particular context used.

1. My recollection of High School Maths reminds me that a "vector quantity" is a physical number having a magnitude (or size) plus a direction, thereby differentiating it from a "scalar quantity" which has size only. An example of each could be velocity (vector) and speed (scalar).

2. The word vector is often used in navigation terminology for aircraft or ships. Often a heading direction is described as "vector 270" for example to suggest "Go West"!

3. The definition provided by the Editor, mentioned previously, was a biological one relating to organisms that transmit germs or disease.

4. A loosely related notion, often used in data processing, describes a vector as a number store of fixed length or as an ordered list of objects. A single dimensioned array in BASIC utilizes this concept and may be called a vector (either row or column) or a list. A two dimensioned entity is termed an array or table and has both rows and columns. A three dimensioned entity may be referred to as a block and has, in addition to rows and columns, layers or planes. These three forms of data arrays are about all that is worth thinking about - although I have worked with scientific programs that have efficiently used seven dimension arrays! The advantage of using arrays lies in the ability to use indexed addressing to refer to a particular cell in the array.

5. All of these uses of vector suggest a movement in a specific direction. Moreover, a starting condition and a final condition is implied. In computer jargon this suggests a type of JUMP instruction (or in BASIC - a GO TO instruction). Vector however does not refer to any-old-JUMP, as a micro-processor is executing this type of instruction often. No, vector is reserved for a more powerful form of JUMP.

VECTOR AS A JUMP.

The technique of "vectoring" generally refers to the ability to intercept ROM based routines and customize other routines or facilities in a microcomputer. In this sense, vectors are merely jumps to machine code routines. Let's digress a little to explain the organization of the ROM and RAM in the VZ computer.

CONTINUED NEXT PAGE

As the VZ executes a BASIC program it branches out of ROM into the Reserved RAM (or Communications Area) located from 7800H to 7AE8H in memory. The Interpreter does this for a number of reasons, one of which is to update the screen display and scan the keyboard. This housekeeping happens 50 times per second and is generated by an interrupt from the Video Display Generator (VDG) chip. Another reason is to permit interception of instructions during program execution. Without this facility the computer would be a fairly inflexible beast if no provision was made for program interception. For example, if it was required to service the keyboard in a special way that hadn't ocurred to the ROM's authors, it would be necessary to implement a USR instruction to call a machine code routine. While this solution would work, it is like using a sledge hammer!

When the VZ initializes, it moves values from ROM into the Reserved RAM, thereby setting up a series of jump vectors for normal operation. Initialization occurs after power-up. Microsoft, the author of the Level II ROM, had enough foresight to provide vectors for most useful routines and checkpoints. There is no doubt this was done for their own use, as later, a Disk BASIC could be easily added by stealing vectors (Who said that a VZ could not be corrupted!). As the vectors reside in RAM they can be changed if desired. The behaviour of routines can be changed in a predictable fashion. Remember a vector is a jump in a prescribed direction.

The versatility and ease of use of vectors is the reason why their insertion, when writing ROM-based routines, is considered good programming practice. The point is that vectors were consciously put there to make this versatility available. Without them in a ROM-based system, one would be stuck with whatever is burned into the ROM. Fortunately for us, vectors are present in the VZ's BASIC ROM Interpreter.

The name given to this seemingly nefarious activity has a ring of felony about it - "vector stealing". It is a delightful description as I thoroughly enjoy every time I steal a vector and achieve an increase in the performance of the VZ!

As an example of the foregoing, I will briefly outline how a very useful "theft" can occur. When an LPRINT token is encountered in a BASIC line, the Interpreter vectors out of the ROM to location 79CAH which is located in the Reserved RAM area. Upon initialization this vector is set to C9H which is machine code for RETURN. The Z80 then vectors back into the ROM-based Interpreter and goes about printing out the following string of text. Herein lies the importance of the vector. The printer routines burnt into the VZ's ROM are designed around the GP-100 type printers. These have largely fallen from favour for Epson-type printers now-a-days. By intercepting the three byte vector located at 79CAH to 79CCH it is possible to redirect the VZ into another routine adapted to the type of printer in use - this is a so-called "printer patch routine" loaded seperately into RAM. A three byte CALL or JUMP instruction can be written into the vector located at 79CAH to cause the redirection. Upon completion of printing the Interpreter returns to ROM again to resume execution and BASIC interpretation.

Pretty neat and useful eh!

Continued on next page

VECTOR THEFT.

Often it is not wise to steal, but to borrow! Imagine that somewhere a keyboard input vector was stolen to facilitate some other form of input. By replacing the vector which "does a jump to keyboard" with a "jump to something else" then the keyboard scanning routine is bypassed. Alternatively both routines could be scanned thereby providing an extra facility. A good example here is the use of the joysticks for input. Whilst the VZ routinely scans for keyboard input, it may be desirable to have the computer also scan for the joystick input ports. (see later).

Within the BASIC Interpreter installed in the VZ quite a few vectors are provided. Usually the vectors point straight back (RETURN) into the ROM routines. When a disk system is connected a number of these vectors are upgraded to provide an enhanced operating system. The problem is to find where these vectors are placed and what their use is. A disassembly of the ROM is required to discover what goes on inside the firmware.

The COPY-PROTECT utility developed by Larry Taylor intercepts a vector located at 79C1H to 79C3H and senses if the BREAK key has been pressed. If so, the BASIC program restarts by jumping to 1B5DH (RUN). This facilty disables listing, changing or copying a BASIC program. The borrowed vector causes a jump to a small machine code routine placed under the BASIC program. The vector at 79C1H is usually called from 032CH which is a routine in ROM used to output to either the screen or printer. This is a very smart use of vector stealing by Larry.

Another use of vector borrowing can be employed when the timing of events is required. As mentioned earlier in this article, the VZ vectors out to the "Interrupt Vector" every 20 msec. The Interrupt Vector is located at 787DH to 787FH and is usually called from 2EBCH or 3E37H in ROM. It is usually set to C9H or RETURN. The interrupt service routine commences at 2EB8H to 2EDAH. The routine at 3E37H to 3E3FH is called upon initialization to set the return vector at 787DH. By borrowing this vector various timing events can be set up.

Interrupts allow external events to get the immediate attention of the CPU. When some event happens, the interrupt rouses the CPU and forces it to carry out certain routines before resuming normal operation. It is a response to an external stimulus.

CONTINUED NEXT ISSUE

P.O.BOX 1972 C.P.O AUCKLAND N.Z

(C) AUCKLAND VZ 300/200 COMPUTER USER'S CLUB 1986

ALL MATERIAL IN THIS NEWSLETTER IS SUBJECT TO COPYRIGHT..
CONTRIBUTED PROGRAMS & ARTICLES CAN BE REPRODUCED ON THE UNDERSTANDING THAT THEY ARE FOR THE PRIVATE USE OF SUBSCRIBERS ONLY.
COPYRIGHT IS ALWAY'S RETAINED BY THE AUTHOR.
THIS CLUB WILL NOT BE HELD RESPONSIBLE FOR THE BREECHING OF COPYRIGHT CAUSED BY THE PUBLICATION OF ANY MATERIAL SENT IN UNDER A FALSE NAME.

YOU ARE WELCOME TO PHONE ME IF YOU HAVE PROBLEMS
PHONE 761-451

DISK LABELS BY DAVE BOYCE . .

```
10 '"DISK LABELS VERS#1 USING VZ-200/300 AND PRINTER/PLOTTER"
20 :
50 CLS:SOUND30,2:PRINT@42,"DISK LABELS":PRINT
60 PRINT"  ENTER NAME OF DISK"
70 INPUT"-:E.G. UTILITY /GAMES /PRINTER  ETC. 8 CHAR'S MAX";IN$
80 INPUT" DISK NO# <PRESS RETURN TO PASS> ";DN$
85 IF DN$="" THEN DN$=".."
90 PRINT:PRINT" DISK NAME IS-:";IN$;" ";DN$
95 INPUT" IS THIS CORRECT <Y/N> ";YN$
96 IF YN$<>"N" THEN 100 ELSE RUN
99 END
100 LPRINTCHR$(18):LPRINT"I":LPRINT"C0,S1"
110 ' BORDER
120 LPRINT"J360,0" 'RIGHT
130 LPRINT"J0,-600" 'DOWN
140 LPRINT"J-360,0" 'LEFT
150 LPRINT"J0,600" 'UP
160 LPRINT"M300,0":LPRINT"I"
170 LPRINT"J0,-600" 'DOWN
180 LPRINT"J-30,0,0,600,-30,0,0,-600,-30,0,0,600"
190 LPRINT"J-30,0,0,-600,-30,0,0,600,-30,0,0,-600"
200 LPRINT"J-30,0,0,600,-30,0,0,-600,-30,0,0,600"
210 LPRINT"R-30,0,0,-300":LPRINT"I"
220 LPRINT"J300,0"
230 LPRINT"R0,300,10,0":LPRINT"I"
240 LPRINT"Q1,S3":LPRINT"P  VZ ";IN$;" DISK NO#.";DN$
250 LPRINT"H"
260 LPRINT"R-33,0":LPRINT"S1":LPRINT"R0,-100"
270 LPRINT"PSIDE A":LPRINT"R0,-240":LPRINT"PSIDE B"
280 LPRINT"R0,-250"
300 LPRINT"C0,S1":LPRINTCHR$(17):END
```

```
10 '"DISK LABELS VERS#2 USING VZ-200/300 AND PRINTER/PLOTTER"
20 :
50 CLS:SOUND30,2:PRINT@42,"DISK LABELS II":PRINT
60 PRINT"  ENTER NAME / TITLE OF DISK"
70 INPUT"-:E.G. UTILITY /GAMES /PRINTER  ETC. 8 CHAR'S MAX";IN$
75 IF LEN(IN$)>8 THEN PRINT"SORRY- TO LONG, TRY AGAIN":GOTO 70
78 IF IN$="" THEN IN$="        "
80 INPUT" DISK NO# <PRESS RETURN TO PASS> ";DN$
85 IF DN$="" THEN DN$=".."
90 PRINT:PRINT" DISK NAME IS-:";IN$;" ";DN$
95 INPUT" IS THIS CORRECT <Y/N> ";YN$
96 IF YN$<>"N" THEN 100 ELSE RUN
99 END
100 LPRINTCHR$(18):LPRINT"I":LPRINT"C0,S1"
110 ' BORDER
120 LPRINT"J360,0" 'RIGHT
130 LPRINT"J0,-600" 'DOWN
140 LPRINT"J-360,0" 'LEFT
150 LPRINT"J0,600" 'UP
160 LPRINT"M300,0":LPRINT"I"
170 LPRINT"J0,-600" 'DOWN
180 LPRINT"J-30,0,0,600,-30,0,0,-600,-30,0,0,600"
190 LPRINT"J-30,0,0,-600,-30,0,0,600,-30,0,0,-600"
200 LPRINT"J-30,0,0,600,-30,0,0,-600,-30,0,0,600"
210 LPRINT"R-30,0,0,-200":LPRINT"I":LPRINT"J300,0"
220 LPRINT"R0,-200":LPRINT"J-300,0"
230 LPRINT"R0,400,310,0":LPRINT"I"
240 LPRINT"Q1,S3":LPRINT"P  VZ ";IN$;" DISK NO#.";DN$
280 LPRINT"R-310,-250"
300 LPRINT"Q0,C0,S1":LPRINTCHR$(17):END
310 ' LINE 78 COULD READ >>       > IF IN$="" THEN IN$="......."
```

DISK LABELS CONT. . . . . .

```
10 '"DISK LABELS VERS#3 USING VZ-200/300 AND PRINTER/PLOTTER"
20 :
50 CLS:SOUND30,2:PRINT@40,"DISK LABELS III":PRINT
60 PRINT"  ENTER NAME / TITLE OF DISK"
70 INPUT"-:E.G. UTILITY /GAMES /PRINTER  ETC. 9 CHAR'S MAX";IN$
75 IF LEN(IN$)>9 THEN PRINT"SORRY- TO LONG, TRY AGAIN":GOTO 70
78 IF IN$="" THEN IN$="        "
80 INPUT" DISK NO# <PRESS RETURN TO PASS> ";DN$
85 IF DN$="" THEN DN$=".."
90 PRINT:PRINT" DISK NAME IS-:";IN$;" ";DN$
95 INPUT" IS THIS CORRECT <Y/N> ";YN$
96 IF YN$<>"N" THEN 100 ELSE RUN
99 END
100 LPRINTCHR$(18):LPRINT"I":LPRINT"C0,S1"
110 ' BORDER
120 LPRINT"J360,0" 'RIGHT
130 LPRINT"J0,-600" 'DOWN
140 LPRINT"J-360,0" 'LEFT
150 LPRINT"J0,600" 'UP
160 LPRINT"R0,-60":LPRINT"J360,0,0,-60,-360,0"
170 ' LINES ACROSS LOOP
180 FOR LF=1 TO 7
190 LPRINT"J0,-30,360,0,0,-30,-360,0"
200 NEXT LF
210 LPRINT"J0,-30,360,0":LPRINT"R-360,0,0,520"
230 LPRINT"I"
240 LPRINT"S3":LPRINT"P  VZ ";IN$
250 LPRINT"H":LPRINT"R0,-60":LPRINT"I"
260 LPRINT"P  DISK NO#.";DN$
280 LPRINT"H":LPRINT"R0,-580"
300 LPRINT"C0,S1":LPRINTCHR$(17):END
310 ' LINE 78 COULD READ >>       > IF IN$="" THEN IN$="......."
```

```
10 '"DISK LABELS VERS#4 USING VZ-200/300 AND PRINTER/PLOTTER"
20 :
50 CLS:SOUND30,2:PRINT@40,"DISK LABELS IV":PRINT
60 PRINT"  ENTER NAME / TITLE OF DISK"
70 INPUT"-:E.G. UTILITY /GAMES /PRINTER  ETC. 8 CHAR'S MAX";IN$
75 IF LEN(IN$)>8 THEN PRINT"SORRY- TO LONG, TRY AGAIN":GOTO 70
78 IF IN$="" THEN IN$="        "
80 INPUT" DISK NO# <PRESS RETURN TO PASS> ";DN$
85 IF DN$="" THEN DN$=".."
90 PRINT:PRINT" DISK NAME IS-:";IN$;" ";DN$
95 INPUT" IS THIS CORRECT <Y/N> ";YN$
96 IF YN$<>"N" THEN 100 ELSE RUN
99 END
100 LPRINTCHR$(18):LPRINT"I":LPRINT"C0,S1"
110 ' BORDER
120 LPRINT"J360,0" 'RIGHT
130 LPRINT"J0,-600" 'DOWN
140 LPRINT"J-360,0" 'LEFT
150 LPRINT"J0,600" 'UP
160 LPRINT"M300,0":LPRINT"I"
170 LPRINT"J0,-600" 'DOWN
180 LPRINT"R0,30" ' UP
190 FOR LF=1 TO 9
200 LPRINT"J-300,0,0,30,300,0,0,30"
210 NEXT LF
220 LPRINT"J-300,0"
230 LPRINT"R300,0,0,30,10,0":LPRINT"I"
240 LPRINT"Q1,S3":LPRINT"P  VZ ";IN$;" DISK NO#.";DN$
280 LPRINT"R-310,-200"
300 LPRINT"Q0,C0,S1":LPRINTCHR$(17):LPRINT:END
310 ' LINE 78 COULD READ >>       > IF IN$="" THEN IN$="......."
```

How to use VDTs and stay healthy

As part of its in-house and on-going office automation programme, Philips NZ Ltd had begun to compile a guidebook for the company's visual display terminal operators.

However, the recent emergence of Repetition Strain Injuiry (RSI) as a major health issue in this country led to production of the booklet "Screen Sense" and a decision to make it and a poster (pictured) freely available to anyone.

To procure this well-intentioned advice, one merely has to write to "Data Systems Division, Philips NZ Ltd, PO Box 2097, Wellington" and ask for it.

Another big employer of VDT operators, Databank Systems Ltd, had recently made a similar effort to keep operators healthy by producing a videotape on RSI symptoms and prevention — the tape available to anyone outside Databank for a nominal production charge of $45.

Below is a sampling of the good advice from "Screen Sense":

• ***Position the keyboard for maximum comfort***

The keyboard should be within reach (around 45cm to 50cm from the body for the average person). You should sit or stand so that your forearms are horizontal while keying and there should be deskspace between you and the keyboard so that you can rest your hands when you are not keying.

• ***Position your hands correctly.***

You should type with your wrists straight (with a straight line through the forearm and middle finger) or bent **slightly**. Bending your wrists too far back while keying places strain on the wrist muscles. You should "drop" your fingers down onto the keyboard rather than reaching back with your hand.

Initially, it may be an advantage to use wrist rests on your desk or workstation if they can be fitted. These allow you to rest your wrists at the correct angle above the keys. After you get used to the posture these can be removed and the position will become automatic.

• ***Press the keys properly.***

Because VDT keys need very little pressure to work, skilled operators can often develop a very light touch compared to a manual typewriter, for example. However, in a manual typewriter when you strike the key, the force you exert with your finger is absorbed by the typewriter in moving the key down. At the end of the key's travel, most of the force you have exerted is gone and your finger "bounces" back as the key rises.

Because of the lighter touch possible with VDT keyboards, there is a temptation to "flick" your fingers on and off the key. By doing this you are actually pressing your finger down and actively lifting if off, and when you lift your finger the force is absorbed not by the keyboard but by the finger itself. The keyboard does not help to bounce your finger up.

It's rather like raising one leg, pushing it down to lightly touch the floor then immediately raising it again. Compare the muscular effort needed to do this with stamping your foot down onto the floor then lifting it... in this case the floor helps bounce your foot up. Stand up and try it!

Try developing a keying action in which you use enough force to depress the key fully (and not skipping over the keys) without pounding on the keyboard (excessive force may damage the keyboard, and tire your fingers).

Developing the right touch will give you the best speed with the least strain, and after a while it will become automatic. It's rather like a pianist, who "strokes" the keys whilst playing.

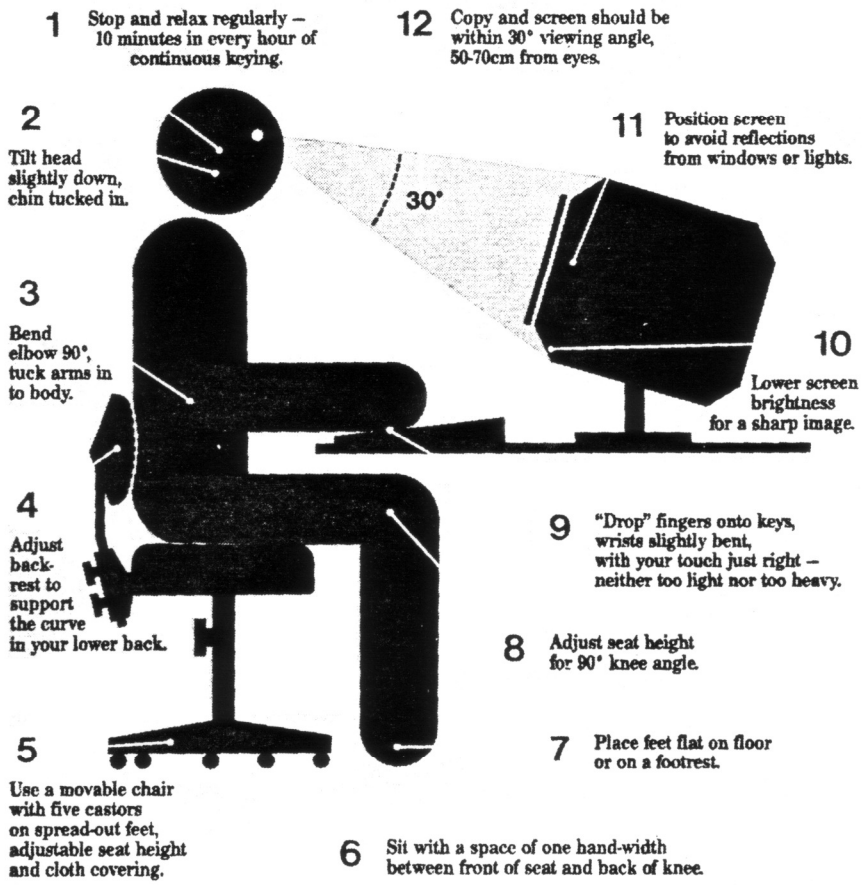
• ***Take a break!***

One of the major contributing factors to keyboard-related health problems is continual use of a VDT without a rest. In some countries laws say that a worker using a VDT must take a short break (often ten minutes) at hourly intervals if using the VDT continually (or five minutes every half hour).

Try to plan your working day so that non-keyboarding tasks which are not urgent can be spread throughout the day to give you short breaks away from the keyboard. You will find the quality of your work will be more uniform as a result.

Here's a useful exercise you can do to alleviate stress in your wrists and fingers. Try doing it when you take a break or when your hands feel tired.

Wrist extension. Hold your arm out from and down from your body at around a 45-degree angle. Stretch the arm out with the palms facing the floor and the fingers outstretched. Firstly move your wrist down as far as you can so that the fingers face the floor, then move your fingers into your palm and try and touch the heel of your hand. Hold this position for a couple of seconds then move the fingers out and straighten your wrist. Repeat the manoeuvre for the other arm.



REPRINTED WITH PERMISSION BITS AND BYTES MAGAZINE